VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Nalezení textu u JPEG obrazu a jeho následná audio syntéza v českém jazyce

# Finding Text in JPEG Image and its Subsequent Audio Synthesis in Czech Language

2013 Dalibor Lis

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání bakalářské práce

Student: **Dalibor Lis**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Nalezení textu u JPEG obrazu a jeho následná audio syntéza v českém jazyce
Finding Text in JPEG Image and its Subsequent Audio Synthesis in Czech Language

Zásady pro vypracování:

Cílem bakalářské práce je návrh a realizace programu pro převod textu obsaženého v souboru JPEG na prostý text a poté syntézu řeči v českém jazyce.

1. Seznamte se s problematikou tvorby JPEG souboru při skenování dokumentu a převod textu v něm na prostý text pro následnou syntézu.
2. Seznamte se s bakalářkou práci pana Bc. A. Brettšnajdra řešící syntézu řeči.
3. Vyberte vhodný program pro převod textu z JPEG souboru na text, tento následně upravte do vhodného tvaru a s použitím existujícího SW nástroje proveďte syntézu řeči v českém jazyce.
4. Navrhnete a realizujte GUI na PC pro ovládání a řízení převodu textu z JPEG souboru na prosti text a řízení syntézy řeči v českém jazyce.
5. Proveďte zhodnocení dosažených výsledků.
6. Vypracujte uživatelskou a programátorskou dokumentaci.

Seznam doporučené odborné literatury:

[1] Milan Sigmund: Zpracování řečových signálů: řešené úlohy a testy, VUT Brno, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 56, 2006, ISBN 8021433000, 9788021433007
[2] Josef Psutka, Luděk Müller, Jindřich Matoušek, Vlasta Radová: Mluvíme s počítačem česky, Academia, 2006 - Počet stran: 746, ISBN 8020013091, 9788020013095

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Lačezar Ličev, CSc.**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013


doc. Dr. Ing. Eduard Sojka
*vedoucí katedry*

prof. RNDr. Václav Snášel, CSc.
*děkan fakulty*

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 16. dubna 2013 .....

Rád bych poděkoval vedoucímu bakalářské práce doc. Ing. Lačezaru Ličevovi, CSc. za cenné rady, připomínky a metodické vedení práce.

## Abstrakt

V bakalářské práci se zabývám problematikou získávání textu z obrazu ve formátu JPEG a následnou audio syntézou získaného textu v českém jazyce. Postup řešení výše uvedené problematiky se dá rozdělit do čtyř částí. V první části je provedena vybraná metoda segmentace na vstupním JPEG obrazu pro zlepšení kvality textu nacházejícím se v obrazu. V druhé části je pomocí metody optického rozpoznávání znaků (OCR) získán text z JPEG obrazu. V třetí části je získaný text upraven do vhodného tvaru pro syntézu, aby nám poskytl co možná nejkvalitnější výsledek. Ve čtvrté části je pak provedena samotná audio syntéza upraveného textu v českém jazyce. V závěru práce vyhodnocuji dosažené výsledky s ohledem na kvalitu výsledné audio syntézy. Aplikace je cílená na slabozraké uživatele, kterým by měla zjednodušit získávání textových informací. Zvolený programovací jazyk pro tuto práci je jazyk Java.

**Klíčová slova:** OCR, segmentace obrazu, JPEG, audio syntéza, Java

## Abstract

In my bachelor thesis I deal with problematic of getting the text from the JPEG image and subsequent audio synthesis of acquired text in Czech language. Procedure for solving the above problems can be separate into four parts. In the first part is performed selected method of segmentation on the input JPEG image for improve quality of text in image. In the second part is extracted text from the JPEG image by using the method of optical character recognition (OCR). In the third part is the acquired text edited into appropriate shape for audio synthesis, to give us the best possible result. In the fourth part is performed itself audio synthesis of edited text in Czech language. In the end of thesis I evaluate acquired results with regard of quality of resulting audio synthesis. The application is targeted for visually impaired users, which should facilitate the acquisition of textual informations. Selected programming language for this work is Java language.

**Keywords:** OCR, image segmentation, JPEG, audio synthesis, Java

## Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| API | – | Application Programming Interface |
| AWT | – | Abstract Window Toolkit |
| DLL | – | Dynamic Link Library |
| DPI | – | Dots Per Inch |
| GUI | – | Graphic User Interface |
| HTTP | – | Hypertext Transfer Protocol |
| JAR | – | Java Archive |
| JFC | – | Java Foundation Classes |
| JNI | – | Java Native Interface |
| JPEG | – | Joint Photographic Experts Group |
| LGPL | – | Lesser General Public License |
| LPC | – | Linear Predictive Coding |
| MHD | – | Městská Hromadná Doprava |
| OCR | – | Optical Character Recognition |
| OS | – | Operating System |
| PDF | – | Portable Document Format |
| RGB | – | Red Green Blue |
| RTF | – | Rich Text Format |
| SO | – | Shared Library |
| SWT | – | Standard Widget Toolkit |
| TTS | – | Text To Speech |
| URL | – | Uniform Resource Locator |
| UTF-8 | – | UCS Transformation Format-8-bit |
| WAVE | – | Waveform audio file format |

# Obsah

# Seznam tabulek

# Seznam obrázků

# Seznam výpisů zdrojového kódu

# 1 Úvod

Problematika získávání textových informací z digitálně pořízených fotografií či naskenovaných dokumentů je v současné době daleko aktuálnější, než tomu bylo před několika lety. Hlavním důvodem je rychlý vývoj elektroniky, převážně vývoj digitální optické elektroniky, jako jsou digitální fotoaparáty či chytré mobilní telefony, které disponují stále kvalitnější optikou. Optika dnešních mobilních telefonů je již mnohdy srovnatelná se staršími digitálními fotoaparáty. Vývoji samozřejmě neunikly ani skenovací zařízení, které také poskytuje kvalitní výstup.

Tento trend má za následek jednak zvýšení kvality pořízených digitálních materiálů, ale také rozšíření této elektroniky mezi větší množství uživatelů. Vzhledem k tomu, že kvalita pořízených materiálů je na poměrně vysoké úrovni, nabízí se možnost zachytit do této podoby textové informace a ty později zpět vyextrahovat. Textové informace v této podobě lze pomocí metody optické rozpoznání znaků z obrazu vyextrahovat a dále s nimi pracovat. Pro běžného uživatele je to velice praktická možnost, jak informace dále zpracovávat. Mezi uživateli však nejsou jen lidé zdraví, ale jsou mezi nimi také uživatelé zrakově postižení, kteří nejsou schopni i přes zvýšenou kvalitu pořízených materiálů plnohodnotně zpracovat informace, které obsahují. Pro takové uživatele je pak možnost vyextrahováni textu z obrazu velice důležitá, protože díky ní si mohou text pomocí audio syntetizéru převést do zvukové podoby, kterou jsou již schopni dále zpracovat.

Náplní mé bakalářské práce je vytvořit takovou aplikaci, která umožní vyextrahování textu z obrazu ve formátu JPEG, který je nejčastějším formátem pořízených obrazů a také umožní tento text pro slabozraké uživatele přehrát pomocí audio syntetizéru. V bakalářské práci je také nutné počítat s tím, že ne vždy musí být vstupní obraz v ideální kvalitě. Pro tyto případy bude možné v aplikaci aplikovat vybranou metodu segmentace obrazu pro zlepšení celkové kvality obrazu tak, aby byl vhodný pro metodu optického rozpoznávání znaků (OCR) a také možnost výběru určité oblasti obrazu pomocí ořezu. Celková podoba uživatelského rozhraní (GUI) bude uzpůsobena pro snadné použití s ohledem na potřeby slabozrakých uživatelů.

# 2 Segmentace obrazu

## 2.1 Úvod do problematiky

Segmentace obrazu je rozsáhlá problematika zabývající digitálním zpracováním obrazu. Nelze říci, že se jedná pouze o jednu samostatnou metodu, ale jedná se zde o rozsáhlé množství metod, které řeší daný problém jiným způsobem. Hlavním účelem segmentace obrazu je rozdělení obrazu do částí, nebo-li lépe řečeno oblastí, které umožní identifikaci objektů v obraze. Častým příkladem identifikace objektů v obraze může být oddělení pozadí od popředí. Jak jsem již výše uvedl, počet metod vedoucích k řešení dané problematiky je více a jsou děleny do několika základních skupin, které pod sebou zaštiťují konkrétní implementace segmentačních metod.[1] Tyto základní skupiny jsou:

- metody založené na segmentaci prahováním
- metody založené na detekci hran (edge-based)
- metody založené na detekci celých oblastí (region-based)
- metody znalostní (knowledge-based)
- metody hybridní

V průběhu segmentace obrazu jsou využívány globální vlastnosti obrazu všemi uvedenými metodami, složitější metody pak vyžadují další výpočty nad obrazem, které umožní přesnější detekci oblastí. Jednou z globálních vlastností obrazu jsou barvy jednotlivých pixelů. Oblast objektu může být detekována na základě společné barvy pixelů, které jsou ve shluku. Častější vyhodnocovanou globální vlastností je jasová úroveň pixelů v obrazu, která je vyhodnocována v podobě histogramu. Na základě znalosti jasové úrovně pixelů je založena segmentační metoda prahování, kterou popisuji v následujících odstavcích. Z další globálních vlastností obrazu můžeme jmenovat body, tvary nebo polohy objektů, které obraz obsahuje. Jmenované vlastnosti jsou v různých segmentačních metodách používány různě a často jsou vyžadovány všechny pro dokonalejší výsledek.[1]

Nemůžeme však opomenout, že i během segmentace mohou nastat problémy. Uvedu tyto problémy s jednoduchými příklady:

- problémy vzniklé v procesu pořizování obrazu - značný šum v obrazu, nerovnoměrné osvětlení
- problémy při analyzování obrazu - obrazová data nejsou jednoznačná, oblasti se překrývají
- nemůžeme říci, že jedna segmentační metoda se hodí na všechny typy obrazů, často je nutné metody kombinovat nebo opakovat s jiným nastavením

Segmentace obrazu slouží často, jako mezikrok pro další zpracování obrazu. Jako příklad mohu uvést mou bakalářskou práci, kde segmentace obrazu slouží pro předpřipravení obrazu pro optické rozpoznávání znaků (OCR).

Informace týkající se úvodu do problematiky segmentace obrazu jsem čerpal z materiálu uvedených na konci práce viz [1].

## 2.2 Segmentace prahováním

Jedná se o jednu z nejednodušších základních metod segmentace, kterou lze dosáhnout dobrých výsledků při aplikování metody s vhodně nastavenými parametry vzhledem k vstupnímu obrazu. Tato metoda také patří k rychlejším metodám segmentace. Základním předpokladem, na kterém je metoda založena je rozdílná odrazivost a pohltivost povrchu oblastí nebo-li objektů, podle kterých jsme schopni oddělit tyto oblasti vzájemně od sebe. Pokud upřesníme vlastnosti které ovlivňují odrazivost a pohltivost povrchu, jedná se o barvy pixelů obrazu a jasové úrovně pixelů obrazu.[1]

Než se dostaneme k detailnímu popisu této metody, je nutné se obeznámit se základním pojmy, které jsou prahová hodnota, nebo-li práh a prahování. V prvním kroku segmentační metody prahováním je nutné určit jasovou úroveň jednotlivých pixelů obrazu na základě jejich tří hodnot barev podle modelu RGB. Takto získané jasové úrovně pixelů zaneseme do grafu. Správný název pro tento graf je histogram. Získali jsme tak přehled nad obsaženými jasovými úrovněmi pixelů v celém obraze.

Prahová hodnota $t$ je zvolená jasová úroveň, podle které lze jednoznačně určit zda je jasová úroveň $x$ libovolného vybraného pixelu obrazu větší ($x > t$), menší ($x < t$) nebo rovna ($x = t$) této hodnotě.

Prahování pak rozumíme, jako procesu převodu vstupního obrazu $p$ na výstupní binární obraz $q$, jehož výstupem získáváme binární obraz, kde body objektu popředí nabývají hodnoty $1$ a body objektu pozadí hodnoty $0$.[1]

Prahování lze také zapsat:

$$q(i,j) = \begin{cases} 1 \text{ pro } p(i,j) \geq t \\ 0 \text{ pro } p(i,j) < t \end{cases}$$

Podle způsobu provedení můžeme prahování rozdělit do více druhů:

- prosté prahování - jedna prahová hodnota je určena pro celý obraz
- použití více prahových hodnot - k přesnějšímu určení oblastí objektů
- adaptivní - prahové hodnoty jsou měněny podle aktuálně zpracovávané části obrazu
- poloprahování - pouze na některé části obrazu je aplikováno prahování

Existuje také více možností, jak můžeme určit prahovou hodnotu:

- z histogramu - nejčastější způsob určení prahové hodnoty
- globální znalost - zde se vychází se všeobecných znalostí, například barvy materiálu a podobně
- experimentálně - časté zkoušení změny prahové hodnoty k docílení lepšího výsledku
- procentní - oblast objektu zaujímá určitou procentní část obrazu, podle jasových hodnot pixelů této procentní oblasti lze určit, pod kterou jasovou úroveň již spadá oblast pozadí

Informace k popisu segmentační metody prahováním jsem čerpal ze zdroje uvedeného na konci práce viz [1].

### 2.2.1 Určení prahové hodnoty z histogramu jasových úrovní pixelů

Určení prahové hodnoty není snadný úkol a často je vybraná hodnota upravována a znovu testována pro získání přesnějšího výsledku. U jednoduchého určení prahové hodnoty se využívá průměr nejtmavější a nejsvětlejší jasové hodnoty z histogramu jasových úrovní jednotlivých pixelů. Tento způsob je při základním určení požadované prahové hodnoty dostačující. Pokud je vyžadována co možná nejpřesnější prahová hodnota, je vhodné využít vybranou statistickou metodu.

#### Stanovení prahu metodou nejmenší chyby

Stanovení prahu metodou nejmenší chyby je založeno na využití minimalizace pravděpodobnosti chybného zařazení prvků obrazové funkce (tzn. že prvek obrazové funkce objektu bude chybně vyhodnocen jako prvek obrazové funkce pozadí a naopak). Jedná se o statistickou metodu, tudíž je nutné uvést základní předpoklady.[2]

$$p(z) = \frac{1}{\sigma\sqrt{2\pi}} exp\left[\frac{-(z-\mu)^2}{2\sigma^2}\right] \qquad (1.1)$$

kde $p(z)$ - normální rozložení hustoty pravděpodobnosti jasových úrovní pixelů objektu

$\mu$ - střední hodnota

$\sigma$ - směrodatná odchylka

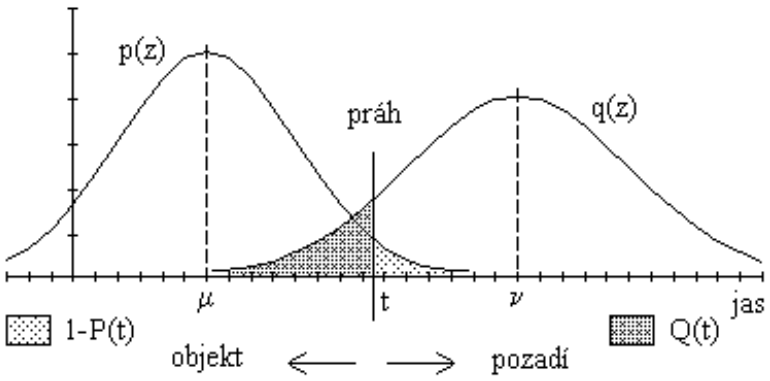
$$q(z) = \frac{1}{\tau\sqrt{2\pi}} exp\left[\frac{-(z-v)^2}{2\tau^2}\right] \qquad (1.2)$$

kde $q(z)$ - normální rozložení hustoty pravděpodobnosti jasových úrovní pixelů pozadí

$v$ - střední hodnota

$\tau$ - směrodatná odchylka

V uvedených dvou vzorcích se dopouštíme jisté nepřesnosti. Hodnoty jasů jednotlivých pixelů jsou nespojité a pouze v omezeném rozsahu. Pokud uvažujeme pixely podle modelu RGB, je tento interval 0 - 255. Dalším předpokladem je, že podíl bodů objektů na obraze je $\theta(0 \leq \theta \leq 1)$. To znamená, že podíl bodů pozadí je $(1 - \theta)$. Počet bodů pozadí bývá z pravidla větší, než počet bodů objektu. Lze tedy předpokládat, že platí $\mu < v$. Hledanou prahovou hodnotu označíme jako $t$. Na obrázku číslo 1 vidíme, kde se prahová hodnota $t$ nalézá.[2]




Obrázek 1: Normální rozložení hustot pravděpodobnosti pro objekty a pozadí s naznačeným prahem t, a pravděpodobnostmi $1 - P(t)$ a $Q(t)$.

Na obrázku se nacházejí dva nové pojmy, které je nutné zadefinovat:

$P(t)$ - je pravděpodobnost, že bod objektu bude vyhodnocen správně jako bod objektu

$P(t)$ získáme vztahem (1.3):

$$P(t) = \int_{-\infty}^{t} p(z)\,\mathrm{d}z \qquad (1.3)$$

$Q(t)$ - je pravděpodobnost, že bod pozadí bude vyhodnocen chybně jako bod objektu

$Q(t)$ získáme vztahem (1.4):

$$Q(t) = \int_{-\infty}^{t} q(z)\,\mathrm{d}z \qquad (1.4)$$

Pravděpodobnost jevu, že bod objektu bude mylně označen, jako bod pozadí je $1 - P(t)$. Také platí, že pravděpodobnost správného určení bodu, jako bodu pozadí je $1 - Q(t)$.[2]

Celková pravděpodobnost chybné detekce odpovídá vztahu (1.5)[2]:

$$\varepsilon = \theta(1 - P(t)) + (1 - \theta)Q(t) \qquad (1.5)$$

Na základě znalosti zadefinovaných pojmů lze vyřešit kvadratickou rovnici (1.6), z níž získáme hledanou prahovou hodnotu $t$.[2]

$$t^2(\tau^2 - \sigma^2) + 2t(\sigma^2 v - \tau^2 \mu) + \tau^2\mu^2 - \sigma^2 v^2 + 2\tau^2\sigma^2 ln\frac{\sigma(1-\theta)}{\theta\tau} = 0 \qquad (1.6)$$

Pro řešitelnost rovnice platí: $\sigma \neq 0, \theta \neq 0, \tau \neq 0, \theta \neq 1$

#### Iterační proces metody nejmenší chyby

Zásadním nedostatkem metody nejmenší chyby popsané výše, je nutnost znát dopředu spoustu informací o zpracovávaném obraze. Řešením je iterační metoda nejmenší chyby. Iterační metoda nejmenší chyby nevyžaduje znalost poměrného zastoupení bodů objektu $\theta$ v obraze. Tato hodnota se zjistí v průběhu iteračního procesu. Nutností je však znalost středních hodnot a hodnot směrodatných odchylek jak objektu tak i pozadí.

Informace k popisu metody nejmenší chyby a iterační metody chyb jsem čerpal ze zdroje uvedeného na konci práce viz [2].

### 2.3 Segmentační metody založené na detekci hran (edge-based)

Následující metody jsou pravděpodobně blíže biologickému rozpoznávání objektů, než je tomu v případě segmentační metody prahováním. Tento způsob segmentace je založen na rychlých změnách obrazové funkce ve vstupním obrazu. Můžeme říci, že takto jsou rozpoznávány objekty pomocí lidského zraku. V případě kvalitního vstupního obrazu lze s vysokou přesností detekovat jednotlivé oblasti objektů a dosáhnout tak velmi dobrého výsledku. Problém ovšem nastává v případech, kdy vstupní obraz není příliš kvalitní nebo v případech, kdy se oblasti překrývají. Horší výsledek tak zapříčiní například značný šum, nerovnoměrné osvětlení, či rozmazání vstupního obrazu. Vzhledem k těmto nedostatkům je předchozí uvedená metoda prahováním účinnější, než metody založené na detekci hran.

Detekce hrany spočívá v nalezení krajních pixelů oblasti, které mají jinou jasovou úroveň, než pozadí, které je za oblastí. Existuje i jiný postup pro nalezení hrany, ve kterém jsou využity hranové operátory. Značnou výhodou při detekci hran je znalost apriorních informací. Apriorní informace jsou informace o objektu, který se nachází v obrazu. Mezi tyto informace se řadí znalost přibližného tvaru objektu nebo například barva objektu. Tyto apriorní informace mohou ve velké míře ovlivnit výsledek segmentace.[1]

Na segmentační metody založené na detekci hran jsou všeobecně kladeny následující nároky:

- minimální nebo žádná chybovost - chybovost chápeme, jako špatně detekovaná místa, která hranami nejsou nebo opomenuté důležité hrany
- přesnost nalezených hran - odchylky mezi hranou nalezenou a hranou skutečnou nesmí být příliš velké, ideálně žádné
- jednoznačnost nalezených hran - jednou detekovaná hrana nemůže být detekována znovu

Uvedené nároky jsou vzhledem k vlastnostem metod dosti vysoké a často dochází k jejich nenaplnění. Problémy, které segmentační metody založené na detekci hran provázejí jsou tak velmi časté.

Mezi tyto problémy patří:

- špatná detekce hrany - hrana je detekována tam, kde žádná hranice mezi oblastmi neprobíhá
- absence detekce hrany - vynechání důležité hrany, tam kde hranice mezi oblastmi dále pokračuje není hrana detekována
- dvojitá detekce hrany - nad hranicí mezi oblastmi dojde k detekci dvou hran

Metod založených na detekci hran je více druhů, mezi časté zástupce patří prahování obrazu hran, sledování hranice nebo Houghova transformace. Jelikož metody založené na detekci hran nejsou svými vlastnostmi příliš vhodné pro řešení problému v zadané bakalářské práci, nebudu je dále rozebírat.

Informace k metodám založených na detekci hran jsem čerpal ze zdroje uvedeném na konci práce viz [1].

## 2.4 Ostatní skupiny segmentačních metod

V úvodu jsem uvedl několik základních skupin segmentačních metod. Detailněji jsem se věnoval prvním dvěma uvedeným, které jsou základem pro všechny ostatní pokročilejší skupiny. Pokročilé skupiny využívají k přesnému určení oblastí neuronové sítě, morfologické operace, štěpení oblastí a další složitější operace. Vzhledem ke své složitosti nejsou pro tuto práci příliš vhodné. Předpokládanými hledanými oblastmi jsou textové znaky, pro které je nejvhodnější řešení metoda segmentace prahováním s možným výběrem konkrétní části obrazu, na které bude aplikována.

# 3 Optické rozpoznávání znaků (OCR)

## 3.1 Úvod do problematiky

OCR nebo-li českým výrazem optické rozpoznávání znaků je technologie, která umožňuje převod textu z klasické podoby na text v digitalizované formě. Předpokladem je, že text v klasické podobě bude naskenován nebo vyfocen pomocí digitálních zařízení typu skener nebo fotoaparát a poté bude v digitalizované formě předložen metodě optického rozpoznávání znaků. V dnešní době technologie optického rozpoznávání znaků podává kvalitní výsledky, avšak nelze jí nazvat za dokonalou. Tak, jako jsem zmiňoval v předcházejících kapitolách problémy, které provázejí segmentační metody, tak i zde mohu tyto problémy znovu uvést, neboť postihují také technologii optického rozpoznávaní znaků. Jedná se převážně o problémy s nekvalitním vstupním obrazem. Slité písmo dohromady, nebo rozmazané písmo je téměř vždy rozeznáno pouze částečně, častěji pak vůbec. Z těchto zkutečností plyne to, že je vždy nutný lidský faktor pro kontrolu a případný přepis špatně rozpoznaného textu. Technologie optického rozpoznávaní znaků je úzce spjatá se metodami segmentace obrazu a to nejen ve společných problémech, ale také v úzké spolupráci v průběhu procesu zpracovávání obrazu. Segmentační metody jsou velmi často využívány technologií optického rozpoznávání znaků k předpřipravení obrazu.

## 3.2 Generace OCR

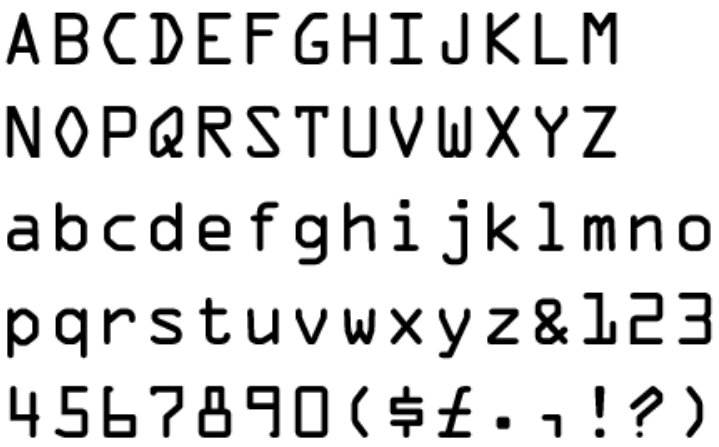
### 3.2.1 První generace

Jedná se zde o úplné počátky (rok 1960) technologie optického rozpoznávání znaků. Vznikali první systémy optického rozpoznávání znaků, které svými vlastnostmi sloužili pouze k základnímu rozpoznávání písma. Aby bylo rozpoznávání úspěšné, byl text psán ve speciálním fontu, který byl těmito systémy snáze rozpoznatelný. Postupem let vývoje se pak začali objevovat i systémy, které dokázali rozpoznávat více druhů fontů. Ručně psané texty nebylo možné rozpoznat. U systému s podporou více fontů bylo rozpoznávání omezeno na použité technologii a také na množství prototypových fontů, podle který docházelo k porovnávání textu.[3]

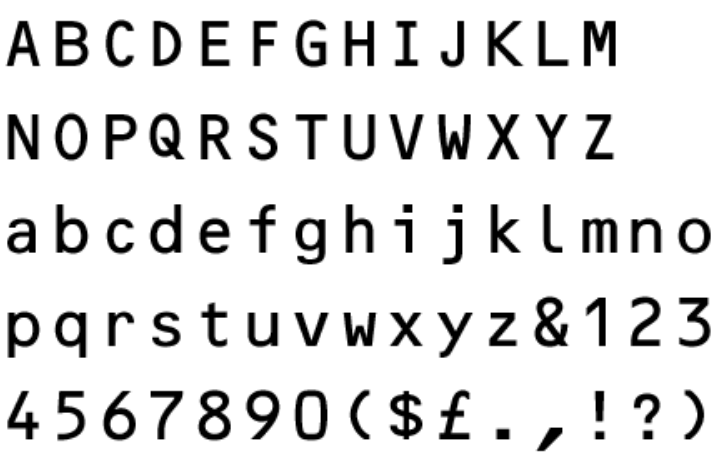
### 3.2.2 Druhá generace

Tato generace je z období let šedesátých až sedmdesátých minulého století. O proti první generaci došlo k značnému pokroku optického rozpoznávání znaků. Samozřejmostí byla podpora více fontů a také bylo umožněno rozpoznávání ručně psaného textu. Ručně psaný text musel být psán s ohledem na požadavky daného OCR systému. V tomto období mnoho firem úspěšně uvádělo na trh nové systémy, které již zmíněné vlastnosti plně využívali. Za zmínku stojí systém IBM 1287 nebo automatický třídič dopisů třídící na základě rozpoznání poštovních směrovacích čísel od firmy Toshiba. Během vývoje OCR systémů byly postupně analyzovány typické požadavky a potřeby, které optické rozpoznávání znaků provázejí. Vyústěním této analýzy byl vznik Amerického standardu OCR fontu OCR-A. Font dle standardu OCR-A se vyznačoval výbornými vlastnostmi

pro optické rozpoznávání znaků, ale také zůstal dobře čitelný pro lidi. Krátce poté byla vytvořena Evropská alternativa v podobě standardu OCR-B. OCR-B standard měl shodné vlastnosti se standardem OCR-A, avšak umožňoval daleko lepší čitelnost pro lidi, než jeho starší alternativa. Nicméně nadcházející nové OCR systémy si poradili s oběma standardy.[3]




Obrázek 2: Ukázka fontu OCR-A




Obrázek 3: Ukázka fontu OCR-B

### 3.2.3 Třetí generace

V třetí generaci docházelo převážně k zlepšování samotného rozpoznávání znaků. Zapracováno bylo převážně na rozpoznávání textu v horší kvalitě a také na ručně psaném

textu. Snahou bylo vytvořit takové systémy, které by nebyly pouze jednoúčelové, jako tomu bylo v předchozích generacích. Ačkoliv byl výkon OCR systému stále větší, a tím i výsledné optické rozpoznávání znaků účinnější, jednoúčelové OCR systémy měli své nenahraditelné zastoupení převážně ve specializovaných činnostech.[3]

### 3.2.4 Čtvrtá generace - současnost

Charakteristikou dnešního stavu OCR systémů je jejich dostupnost pro širší veřejnost a také posun od úzce specializovaného odvětví k běžnému použití v rozmanitých druzích oborů. Cenově jsou OCR systémy taktéž dostupnější, avšak ty opravdu kvalitní jsou finančně nákladnější. V současnosti jsou OCR systémy implementovány v podobě softwaru pro počítače tak, aby byly co nejvíce dostupné.[3]

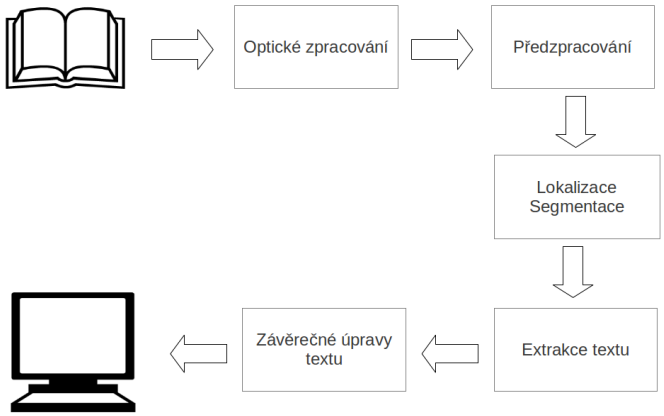
## 3.3 Učební proces

Aby každý systém optického rozpoznávání znaků dokázal určit, o jaký znak ve vstupním obrazu se jedná, je nutné, aby systém předem znal možné případy znaků, které bude v budoucnu zpracovávat. Fáze, ve které se systém optického rozpoznávání znaků učí budoucí zpracovávané případy se nazývá fáze tréningu nebo-li učení. Fázi tréningu lze popsat, jako předložení obrazců znaků, jako jsou písmena, interpunkce nebo jiné speciální znaky, ke kterým je jednoznačně systému určeno, co daný znak symbolizuje. Vhodné je stejné znaky OCR systém učit ve více druzích možných interpretacích. Rozdílné interpretace stejného znaku nazýváme třídy znaku. Proces porovnávání znaku ze vstupního obrazu spočívá v porovnávání vlastností znaku s již předem známými třídami znaků. Tam, kde dochází k největší schodě vstupního znaku se třídou znaku, je určena daná třída jako odpovídající vstupnímu znaku. Učení systému optického rozpoznávání znaků je zdlouhavý proces, kdy kvalita systému v rozpoznávání je určena délkou a množstvím naučených případů znaků, které budou v budoucnu zpracovávány. Teoreticky nic nebrání tomu, aby se OCR systém naučil rozpoznávat psaný text nebo i jiné složitější případy znaků. Problém však nastává v ne vždy stejné interpretaci zápisu daného znaku zejména v případě ručně psaného textu. Celý tento proces vzhledem ke své časové náročnosti probíhá již ve fázi vývoje OCR systému a tak nejsou koncoví uživatelé tímto procesem zatěžováni.

## 3.4 Části OCR systému

Nelze říci, že se v OCR systému jedná pouze o část rozpoznávání znaků. I když je to jedna z důležitých částí, není to část jediná. Každý OCR systém se skládá s více částí, které vzájemně spolupracují v průběhu celého procesu rozpoznávání znaků. Cílem všech těchto částí je dosažení maximálně kvalitního rozpoznání všech znaků ve vstupním obraze. Ačkoliv je dnes výstup OCR systému po provedení všech zmíněných částí poměrně kvalitní, nelze nikdy spoléhat na dokonalý výsledek a kontrola člověkem je nutná. Mezi jednotlivé části patří optické zpracování, kterým získáme digitální podobu vstupního obrazu, segmentace a lokalizace digitálního obrazu, které slouží převážně k odstranění členitostí v obrazu, dále pak předzpracování obrazu, kde dochází k odstranění šumu v obrazu,

extrahování samotného textu podle tříd znaků a na závěr opravy špatně rozpoznaných znaků, kde už se počítá i s přispěním lidského faktoru. Průběh zpracování obrazu OCR systémem můžeme vidět na obrázku číslo 4. [3]




Obrázek 4: Průběh zpracování obrazu OCR systémem

## 3.5 Seznámení s vybranými částmi OCR systému

### 3.5.1 Předzpracování

V části, která se zabývá předzpracováním obrazu počítáme s již vzniklým digitálním obrazem, který byl pořízen digitální optickou elektronikou, jako je skener nebo digitální fotoaparát. Tento obraz se dále převádí do ekvivalentního binárního obrazu. Binární obraz jsem detailněji popisoval v části věnované segmentační metodě prahováním. Pro připomenutí, binární obraz odpovídá počtem bodů a velikostí matice svému digitálnímu předchůdci, avšak na místo barev jednotlivých pixelů má každý pixel binární hodnotu nula nebo jedna, podle které lze určit, zda se jedná o popředí nebo o pozadí. Součástí převodu digitálního obrazu na binární obraz je odstranění rušivých elementů, jako je šum v obrazu, rozpadlé nebo neúplné znaky a další nežádoucí elementy. V případě rozpadlých znaků se používá více způsobů řešení tohoto problému pomocí aplikování filtrů. Častým filtrem je filtr vyhlazení. Ten lze rozdělit na dva základní typy:

- Vyplňování
- Ztenčování

Vyplňování postupuje následujícím postupem. V případě, že znak obsahuje prázdné body, není tedy celistvý, jsou tyto body automaticky vyplněny odpovídajícím typem bodu,

jakým je tvořen samotný znak. Ztenčování pracuje na opačném principu. V případě nalezení prázdných bodů ve znaku je celý znak zúžen až do míst chybějících bodů. Nedílnou součástí předzpracování je taktéž normalizace znaků, která má za úkol upravit znaky tak, aby měli stejný sklon a taktéž rotaci a jednotkovou velikost.[3]

### 3.5.2 Lokalizace a segmentace

Lokalizace a segmentace úzce navazují na předzpracování digitálního obrazu, protože získané informace z předchozí části využívají během své části práce. Úkolem lokalizace a segmentace je určení rozložení textu v obraze. Jelikož v předchozí části díky tvorbě binárního obrazu vznikl i histogram jasových úrovní jednotlivých pixelů, je tento histogram znovu využit v detekci řádků a jednotlivých textových znaků. Musím upozornit, že se zde nejedná o přímé určení třídy znaku, tedy finální určení o jaký znak se jedná, ale úkolem je samotná detekce umístění neurčitého znaku vzhledem k celkovému obrazu. Taktéž je nutné zohlednit možnou grafiku, která je vedlejším doplňkem obrazu, jako například vložená fotografie do textu. Pokud v takové vložené fotografii není umístěn detekovatelný text, měla by být zcela ignorována.[3]

### 3.5.3 Extrakce textu - extrakce příznaků

V této části dochází k nejzásadnějšímu a zároveň nejproblematičtějšímu úkolu OCR systému. Dochází zde k určení přesných hodnot již nalezených znaků z předchozí části. Pro řešení tohoto úkolu existuje více druhů metod, které k problému přistupují odlišně. V zásadě můžeme tvrdit, že souhrnně existují dvě typové skupiny:

- **přímé určení hodnoty znaku** - skupina metod využívající k porovnání extrahované příznaky znaku, které porovnává s natrénovanými (naučenými) třídami znaku k určení jeho hodnoty
- **získávání specifických rysů znaku** - skupina metod neurčující hodnotu znaku přímo, ale provádějící dodatečné výpočty nad znakem k určení jeho hodnoty

#### Přímé určení hodnoty znaku

Skupina metod založených na přímém určení hodnoty znaku pracuje se znakem jako s rozloženými body v mřížce. Lépe řečeno pracuje s maticí bodů, na které provádí porovnávání vybraných vlastností (extrahovaných příznaků) s třídami znaku. Vzhledem k povaze maticového zápisu znaku tak lze provádět více způsobů jeho detekce. Uvedu dva základní způsoby.

#### Přímé určení hodnoty znaku - Rozdělení do pásem

Způsob rozdělení do pásem pracuje s rozdělením nalezeného znaku do jednotlivých pásem nebo-li oblastí, kde v každé oblasti dochází ke zkoumání a porovnávání histogramu tmavých bodů s třídami znaku získaných z fáze tréningu. Ukázku tohoto způsobu můžeme vidět na následujícím obrázku číslo 5.[3]

Obrázek 5: Rozdělení znaku do pásem

#### Přímé určení hodnoty znaku - Průsečíky vektorů

Stejně, jako způsob rozdělení znaku do pásem pracuje i způsob průsečíků vektorů s maticovým zápisem znaku. OCR systém si předem zvolí vektory, u který následně zkoumá počet průsečíků těchto vektorů. Názorněji to lze ukázat na následujícím obrázku. Čím více společných průsečíků má analyzovaný znak s třídou znaku tím je větší přesnost určení správné hodnoty znaku.[3]

Obrázek 6: Průsečíky vektorů

#### Získávání specifických rysů znaku

Souhrnně se pro tento způsob detekce znaku používá název strukturální analýza. Strukturální analýza je daleko více náročnější co do provedení i co do časové náročnosti. Znaky jsou popisovány geometrickou a topologickou strukturou. Implementačně je metoda velmi náročná a zatím je více ve fázi výzkumu. Velkou výhodou je tolerance šumu, který nemá takový vliv na špatné určení hodnoty znaku, jako je tomu v případě metod založených na přímém určení hodnoty znaku.[3]

### 3.5.4 Závěrečné úpravy textu

Závěrečné úpravy textu nejsou u systémů optického rozpoznávání znaků nutně povinným krokem, ovšem jak již bylo několikrát uvedeno, je tento krok nutný k dosažení přesného vyextrahování textu. Systém může již podle naučených pravidel z předchozích úprav textu znát vybrané znaky nebo slovní spojení, která nahrazuje za zvolený vzor. I přesto, že systém provede tuto závěrečnou část úprav, nelze očekávat 100% úspěšnost. Proto je zásah člověka nevyhnutelný a očekává se ještě korektura textu právě od něj.

Informace týkající se metod OCR systémů a jejich detailnějšího popisu jsem čerpal ze zdroje uvedeného na konci práce viz [3].

## 3.6 Přehled dostupných OCR systémů

Aktuální nabídka OCR systémů je poměrně pestrá, avšak ve většině případů si každý OCR systém nese svá omezení. Tato omezení spočívají převážně v kvalitě optického rozpoznání textu předloženého v obrazu a způsobu ovládání OCR systému. Ačkoliv ještě před několika lety neexistovalo použitelné bezplatné řešení OCR systému, dnes se již situace zlepšila. Stále existují placená řešení OCR systémů, ale naproti tomu existují i řešení bezplatná a některé i s otevřeným zdrojovým kódem (open source). Uvedu zde základní zástupce placených řešení a zástupce řešení bezplatných.

- Placená řešení
  - ABBYY Cloud OCR SDK
  - Asprise OCR
  - ExperVision TypeReader RTK
- Bezplatná řešení
  - Tesseract-OCR
  - OnlineOCR.net
  - GOCR

V předcházejícím výčtu OCR systémů jsem uvedl nejznámější zástupce. Ve výčtu se vyskytují OCR systémy v podobě webových služeb, klasických webových aplikací, zdrojových kódů, či spustitelného softwaru. Je nutné uvést, jaké omezení si OCR systémy implementované v různých podobách sebou nesou.

Začněme webovými službami. Řešení v podobě webové služby nemá bezplatného zástupce, což je hlavním nedostatkem. Způsob komunikace se službou je díky již předem přichystaným API snadný, avšak aplikace vyžaduje připojení k síti internet k provedení svého úkolu.

Komunikace s OCR systémy skrze webové rozhraní je nepoužitelná v případě implementace vlastní aplikace, jako je tomu v této bakalářské práci. Existuje možnost cíleného zasílání HTTP požadavků na webový server, avšak odpovědi serveru jsou neočekávané a aplikace postavená pouze na této komunikaci nemá zajištěnu funkčnost v případě změn na straně poskytovatele online OCR webu. Platí zde i další omezení, jako v případě webových služeb a to nutnost neustálého připojení k síti internet.

Zdrojové kódy OCR systému, které poskytuje projekt GOCR si nesou omezení v podobě výsledného rozpoznání znaků. Výsledky rozpoznávání jsou nedostatečné pro nasazení do aplikace spoléhající na kvalitu výsledku OCR systému. Tento problém je zapříčiněn nedostatkem tréningových dat.

Nejvhodnější řešením se jeví využití bezplatného OCR systému Tesseract-OCR, který je již sestaven včetně tréningových dat. Za vývojem Tesseract-OCR stojí firma Google a v počátcích vývoje i firma HP. Systém je napsán v jazyku C++ z důvodů dosažení maximální rychlosti optického rozpoznávání znaků. Tesseract-OCR podává velice uspokojivé výsledky rozpoznávání textu v obraze a proto jsem jej zvolil, jako OCR systém vhodný

pro implementaci aplikace této bakalářské práce. Nad OCR systémem Tesseract-OCR je nutné sestavit vhodné API tak, aby mohl být začleněn do implementace výsledné aplikace. Detailněji tento OCR systém popisuji v části věnované softwarovým prostředkům použitých pro vývoj aplikace.

# 4 Audio syntéza

## 4.1 Úvod do problematiky

Audio syntézu chápeme, jako úkol pro počítač, který na základě předloženého textu vyprodukuje odpovídající audio výstup. Je důležité poznamenat, že vyprodukovaný audio výstup počítač předem nezná a vytváří jej dynamicky. Audio syntézu lze provést více způsoby. Jednodušším příkladem audio syntézy je skládání audio souborů (nejčastěji ve formátu WAVE) do jednoho celku s cílem vyprodukovat vybraný souvislý zvukový výstup. S tímto příkladem se můžeme setkat v MHD, na vlakových či autobusových zastávkách nebo na letištích, kde takto vznikají pokyny pro cestující. Složitějším příkladem je pak audio syntéza textového vstupu (TTS), kde zvukový výstup vzniká na základě jednotlivých znaků a slovních spojení v textu. Abychom pochopili, jak pracuje počítač v průběhu audio syntézy, je nutné obeznámit se základními vlastnostmi tvorby řeči lidskými orgány.[4]

## 4.2 Hlasové ústrojí člověka - vznik řeči

Proces vytváření lidské řeči je složitý akt, kde spolu vzájemně spolupracují lidské orgány, z nichž každý orgán provádí vybranou část procesu s cílem vytvořit řeč. Na počátku procesu tvorby řeči jsou plíce. Plíce tvoří hlavní zdroj energie, který je dále zpracováván ostatními orgány. Na základě velikosti tlaku vzduchu, který plíce vytvoří je ovlivněna intenzita hlasu. Vzduch je dále přenesen přes hrtan k hlasivkám. Zde dochází k modulaci energie. Hlasivky nabývají dvou stavů a to zavřené nebo otevřené. Pokud jsou hlasivky otevřené, je produkován šum. Pokud hlasivky kmitají, vzniká periodický signál. Pro člověka dle pohlaví i věku jsou definovány typické hodnoty základního tónu řeči. Hodnoty jsou uvedeny v následující tabulce číslo 1.[5]

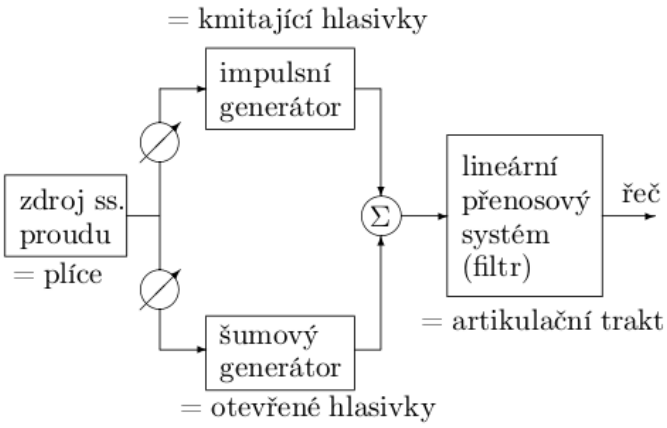
| | |
|---|---|
| muži | 90 - 120 Hz |
| ženy | 150 - 300 Hz |
| děti | 350 - 400 Hz |

Tabulka 1: Typické hodnoty základního tónu řeči

Z hlasivek postupuje zvuk do hlasového traktu. Tento trakt také nazýváme traktem artikulačním nebo modifikačním ústrojím. Složení artikulačního traktu se skládá z hltanu (pharynx), měkkého patra (vélum), jazyka, ústní dutiny, nosní dutiny, zubů a rtů. Schématický model tvorby lidské řeči v hlasovém traktu lze znázornit obrázkem číslo 7.[5]

## 4.3 Specifické rysy pro český jazyk

Český jazyk obsahuje specifické rysy v mluvené řeči, které se projevují v akustickém spektru výsledného zvuku. Mezi tyto specifické rysy patří souhlásky, samohlásky, jejich znělé či neznělé formy a podobně. V akustickém spektru je obsažen základní tón hlasu a




Obrázek 7: Model hlasového ústrojí[4]

také všechny ostatní vyšší hodnoty frekvencí, které vznikají úpravou hlasovým traktem, který mění rezonanci. Věda, zabývající se změnami základního tónu hlasu a jeho odvozenými variantami se nazývá fonetika. Fonetika souhrnně nazývá základní tón hlasu a jeho všechny ostatní vyšší hodnoty frekvencí jako formanty. Formanty jsou označovány čísly od 0 do $n$ ve tvaru $F_0, F_1, F_2, F_3, ..., F_n$, kde $F_0$, odpovídá základnímu tónu lidské řeči. [5]

### 4.3.1 Samohlásky

Pro samohlásky jsou z pohledu fonetiky důležité formanty $F_1$ a $F_2$. Tyto formanty jsou typické pro český jazyk, avšak v jiných jazycích je zkoumán i formant $F_3$. Samohlásky vznikají při pozvolném průchodu vzduchu hlasovým traktem. Znamená to tedy, že samohlásky jsou čisté tóny bez šumů. Samohlásky dělíme dále na znělé a neznělé. Častější jsou samohlásky znělé, kde při jejich tvorbě pracuje artikulační ústrojí (trakt). Rezonanci znělých samohlásek ovlivňuje dutina ústní, dutina nosní, dále pak měkké patro, čelisti a v neposlední řadě rty. Přehled hodnot frekvencí odpovídající českým samohláskám prvních dvou formantů můžeme vidět v tabulce číslo 2. [5]

| **samohláska** | $F_1$ | $F_2$ |
|---|---|---|
| u | 300 - 500 Hz | 600 - 1000 Hz |
| o | 500 - 700 Hz | 900 - 1200 Hz |
| a | 750 - 1100 Hz | 1100 - 1500 Hz |
| e | 500 - 700 Hz | 1500 - 2000 Hz |
| i | 300 - 500 Hz | 2000 - 3000 Hz |

Tabulka 2: Hodnoty formantů $F_1$ a $F_2$ pro české samohlásky

### 4.3.2 Souhlásky

Souhlásky se odlišují od samohlásek hlavně v přítomností šumu ve výstupním tónu. Artikulační trakt, převážně jazyk, zuby a rty představují překážky, které vytvářejí rezonanci. Princip vytvoření šumu spočívá v hromadění vzduchu před překážkou, kterou představují uvedené artikulační orgány a následné uvolnění nahromaděného vzduchu. Uvolnění nahromaděného vzduchu se nazývá exploze. Takový způsob šumu je vytvořen pomocí úplné překážky. Vybrané české souhlásky však vyžadují i specifičtější druh šumu. Aby bylo možné změnit intenzitu šumu, hlasový trakt využívá polopropustné překážky. Při průchodu určité části vzduchu přes polopropustnou překážku vzniká specifický druh šumu. České souhlásky jsou uvedeny podle druhu překážky při jejich vzniku v tabulce číslo 3.[5]

| **typ překážky** | **souhlásky** |
|---|---|
| úplná | p, t, ť, k, b, d, ď, g, m, n, ň |
| polopropustná | s, š, f, ch, z, ž, v, h, l, j, r, ř |
| kombinace obou | c, č, dz, dž |

Tabulka 3: České souhlásky s typem překážky při jejich vzniku

Pro upřesnění, pokud je při vzniku souhlásky přítomna úplná překážka, jedná se o souhlásku závěrovou. V případě přítomnosti polopropustné překážky se jedná o souhlásku úžinovou a v případě kombinace obou druhů překážek se jedná o polozávěrovou souhlásku. Dalším specifickým prvkem českých souhlásek je jejich znělost. Existují dvě skupiny z pohledu dělení podle znělosti a sice znělé a neznělé souhlásky. Rozdíl mezi znělou souhláskou a neznělou souhláskou spočívá v přítomnosti základního tónu hlasu. Pokud se bavíme o neznělé souhlásce, tak v takové souhlásce není obsažen základní tón hlasu. Vzduch proudí volně hlasovým ústrojím, hlasivky jsou od sebe oddáleny a tudíž nepředstavují žádnou překážku, která by vedla ke vzniku rezonance. Tento stav se dá přirovnat k běžnému dýchání. Naproti tomu, pokud se bavíme o znělých souhláskách, tak to jsou takové souhlásky, které základní tón hlasu obsahují. Hlasivky tentokrát způsobují svým mírným sevřením rezonanci vzduchu a tím způsobí přidání základního tónu do výstupního zvuku. Znělost a neznělost souhlásek má často mezi sebou vazbu. Značná část českých souhlásek má svou znělou variantu i neznělou variantu a tvoří tak pár (párové). V případě souhlásek, které mají svou znělou i neznělou variantu rozhoduje o finálním výstupním zvuku artikulační ústrojí. Artikulace je vždy u takové souhlásky stejná, avšak přítomnost základního tónu řeči změní výstupní zvuk. Přehled všech českých souhlásek včetně jejich znělosti je uveden v tabulce číslo 4.[5]

## 4.4 Metody syntézy řeči na počítači

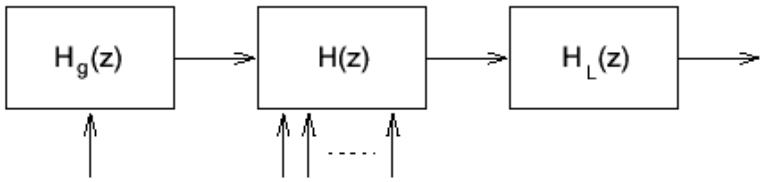
### 4.4.1 Formantová syntéza

Jedná se o jeden ze dvou druhů zástupců řešících audio syntézu ve frekvenční oblasti, který lze realizovat na počítači formou software. Formantová syntéza přistupuje k pro-

| Souhlásky | | Závěrové | Úžinové | Polozávěrové |
|---|---|---|---|---|
| **Párové** | neznělé | p, t, ť, k | s, š, f, ch | c, č |
| | znělé | b, d, ď, g | z, ž, v, h | dz, dž |
| **Nepárové** | znělé | m, n, ň | l, j, r, ř | |

Tabulka 4: Přehled českých souhlásek

blému způsobem nastavování jednotlivých parametrů, tak jako by se jednalo o lidský hlasový trakt. Díky tomu je způsobem tvorby zvuku prakticky nejblíže k reálnému vzniku zvuku v lidském hlasovém traktu, jak je popsáno výše. Je nutné však poznamenat, že formantové syntetizéry nemodelují lidský hlasový trakt úplně. Principem je rozdělení zpracovávání zvuku na části, konkrétně tři, kde každá část vykonává úlohu, která odpovídá zjednodušenému vzniku zvuku v lidském traktu. Chování formantového syntetizéru je následující. Zdroj vzniku zvuku je popsán ve frekvenční oblasti, jako spektrum $H_g(z)$, zdroj dále budí hlasový trakt s přenosovou funkcí $H(z)$, který lze popsat, jako skupinu rezonátorů. Výsledná podoba zvuku je pak dána modelem vyzařování řečového signálu $H_L(z)$. Celý popsaný postup vzniku audio syntézy pomocí formantového syntetizéru můžeme vidět na obrázku číslo 8.[6]




Obrázek 8: Model formantového syntetizéru

### 4.4.2 Syntéza na bázi lineární predikce - LPC

Další ze zástupců řešících audio syntézu ve frekvenčním spektru. Hlavním rozdílem mezi formantovými syntetizéry a syntetizéry založenými na bázi lineární predikce je, že se syntetizéry založené na bázi lineární predikce snaží modelovat přenosovou charakteristiku hlasového traktu najednou a nerozdělují ji tak do částí, jako formantové syntetizéry. LPC syntetizér bývá realizován častěji, protože lze popsat soustavami lineárních rovnic a lze jím dosáhnout při vhodně zvolených filtrech velmi věrné reprodukce lidského hlasu.[5][6]

# 5 Softwarové řešení aplikace

## 5.1 Programovací jazyk Java

Programovací jazyk Java je objektově orientovaný programovací jazyk, který vychází koncepčně z jazyku C++. Jazyk Java se však zbavuje negativních vlastností jazyka C++, které mohou při nevhodném použití způsobovat pády programu nebo systému. Mezi tyto negativní vlastnosti patří práce s ukazateli na paměť, které jsou při špatném uvolňování viníky úniků paměti a následným vyústěním výše uvedených problémů. Jazyk Java používá pro práci s objektovými datovými typy odkazy nebo-li reference, o jejichž uvolňování se stará mechanismus známý, jako Garbage Collector. Dále je důležité poznamenat, že jazyk Java běží v tzv. virtuálním stroji, což jazyku umožňuje nezávislost na platformě. Lze jej tedy provozovat pod všemi běžnými OS (Windows, Linux, MAC OS), kde existuje implementace virtuálního běhového prostředí. Další výhodou jazyku Java je podpora velkého množství knihoven, které jsou standardní součástí a není je nutné externě dohrávat.

Jazyk Java poskytuje pro tvorbu uživatelského rozhraní (dále GUI) několik možností. První možností je použití původní implementace GUI nazvané AWT. Tato možnost je již ovšem nedoporučovaná, protože je zastaralá. Doporučenou možností je použití standardní grafické knihovny JFC, známější pod názvem Swing. Grafickou knihovnu Swing jsem zvolil pro implementaci GUI aplikace. Z historických důvodů byla vyvinuta knihovna SWT, která měla vylepšit nedostatky knihovny Swing. Ve výsledku však nenabízí žádné zásadní výhody o proti Swingu. Doplňující možností je použití grafické knihovny Qt, která je určená pro vývoj GUI v C++ programech, ale umožňuje i použití v programech psaných v jazyku Java.[7]

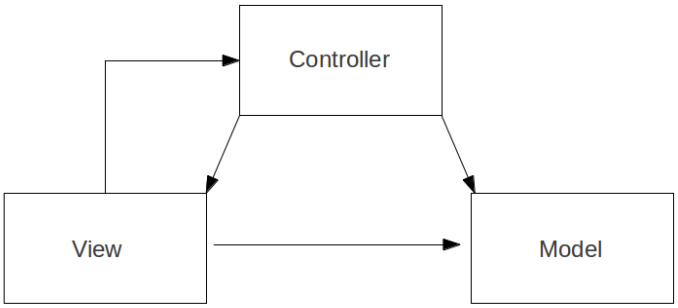
## 5.2 Architektura aplikace

Aplikace je navržena podle návrhového vzoru Model View Controller (dále MVC). Předností tohoto návrhového vzoru je rozdělení aplikace do tří logických částí, které jsou chápány, jako tři samostatné celky. Výhodou tohoto rozdělení je oddělení kritických částí kódu od implementace vzhledu a také k celkovému zpřehlednění kódu. Aplikace napsané podle vzoru MVC jsou snadnější pro údržbu a úpravy kódu. Případní noví programátoři se v kódu rychleji zorientují a pochopí souvislosti. Rozdělení aplikace lze vidět na obrázku číslo 9.

V části Model je řešena kompletní logika aplikace. To znamená, že obsahuje metody pro aplikování segmentace obrazu, ovládání OCR systému, spouštění audio syntetizéru a další metody pracující s logikou aplikace.

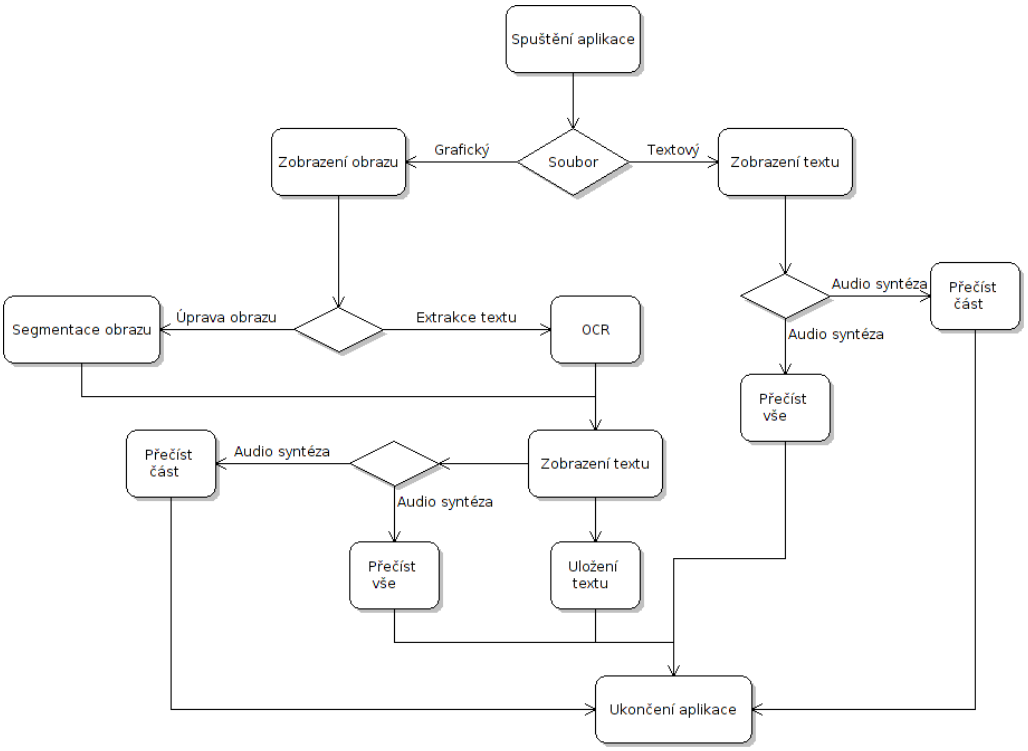
V části View je implementován kompletní vzhled aplikace. Samotná část View není schopna vykonávat žádnou logickou operaci aplikace, slouží pouze, jako uživatelský ovládací prvek, kterým je uživatel schopný ovlivnit chování aplikace. Aby aplikace mohla správně inicializovat vzhled, přistupuje View k modelu přímo, ale pouze k získání inicializačních hodnot. Veškeré další reakce z uživatelského rozhraní jsou s modelem propojeny skrze část Controller.

V poslední části nazvané Controller jsou řešeny události z uživatelského rozhraní a jejich propojení na logické operace modelu.




Obrázek 9: Návrhový vzor Model View Controller (MVC)

Vývojový diagram aplikace lze pak vidět na obrázku číslo 10.




Obrázek 10: Vývojový diagram aplikace

## 5.3 Aplikování segmentační metody

V aplikaci je použita segmentační metoda prahování obrazu (výpis číslo 1). Tato metoda je vhledem k své povaze vhodná k účelu aplikace a poskytuje dobré výsledky. Prahová hodnota je aplikována na celý obraz. Metoda je volána na základě uživatelské žádosti a není tudíž striktně aplikována při každém volání metody optického rozpoznávání znaků. Aplikována je na celý aktuálně zobrazený obraz a na základě vybrané prahové hodnoty odděluje popředí od pozadí.

Vhodnou prahovou hodnotu propočítává aplikace pomocí metody uvedené ve výpisu číslo 2 pro každý zobrazený obraz zvlášť a nabízí jej uživateli k použití. Uživatel nicméně může nastavit vlastní parametr prahové hodnoty a ten bude při aplikování metody použit. Pokud se uživatelsky nastavený parametr prahové hodnoty ukáže, jako nejlepší, je možnost tento parametr uložit do souboru, který bude vytvořen vedle souboru zpracovávaného obrazu a při příštím načtení obrazu bude automaticky tato hodnota načtena.

```
public void transform(int threshold) {
    if (image != null) {
        int imageWidth = image.getWidth();
        int imageHeight = image.getHeight();

        for (int y = 0; y < imageHeight; y++)
            for (int x = 0; x < imageWidth; x++) {

                int pixelColor = image.getRGB(x, y);
                int red = (pixelColor & 0x00ff0000) >> 16;
                int green = (pixelColor & 0x0000ff00) >> 8;
                int blue = pixelColor & 0x000000ff;

                int avg = (red + green + blue) / 3;
                if (avg < threshold) {
                    image.setRGB(x, y, 0x00000000);
                } else {
                    image.setRGB(x, y, 0xFFFFFFFF);

                }
            }
    }
}
```

Výpis 1: Volání metody pro aplikování metody prahování

```
public double getThresholdValue(File file) throws IOException {
    return new ThresholderCore(file).getThresholdValue();
}
```

Výpis 2: Volání metody výpočtu prahové hodnoty obrazu

## 5.4 Výběr určité části obrazu

V aplikaci jsem vytvořil třídu *PictureViewer*, která svými metodami slouží k ořezání obrazu a získání tak jen vybrané části. Účel této třídy je zřejmý, třída umožňuje odstranit přebytečné části obrazu, které vedou k nepřesnému určení prahové hodnoty a dále pak k nekvalitnímu rozpoznání znaků metodou OCR. Ukázku výňatku kódu řešícího ořez obrazu můžeme vidět ve výpisu číslo 3.

```
int x1 = selection.x;
int y1 = selection.y;
int width = selection.width;
int height = selection.height;

int absoluteWidth = width + x1;
int absoluteHeight = height + y1;

if (x1 <= iconWidth && x1 >= 0 && y1 <= iconHeight && y1 >= 0
    && absoluteWidth <= iconWidth
    && absoluteHeight <= iconHeight && width > 0
    && height > 0) {
  BufferedImage bImage = getBufferedImage();
  if (bImage != null) {
    bImage = bImage.getSubimage(x1, y1, width, height);
    setIcon(new ImageIcon(bImage));
    fireIconListeners(bImage);
  }
}
selection = null;
repaint();
```

Výpis 3: Ukázka výňatku metody řešící ořez obrazu

K této třídě jsem vytvořil sadu tříd a rozhraní, které jsou navrženy podle návrhového vzoru Observer (Pozorovatel). Tyto třídy slouží k provázání událostí vytvořených třídou PictureViewer s logikou aplikace (modelem aplikace). Třídu *PictureViewer* lze tak použít i v budoucnu v jiných aplikacích, kde pomocí uvedených tříd a rozhraní lze reagovat na události, které třída *PictureViewer* při ořezu obrazu vyvolá. Třída se aplikuje do uživatelského rozhraní stejně, jako standardní Java komponenty grafické knihovny Swing.

## 5.5 Aplikování metody optického rozpoznávání znaků (OCR)

Zvoleným OCR systémem pro aplikaci byl OCR systém firmy Google s názvem Tesseract-OCR. Tento systém je poskytován pod licencí open-source a podporuje více jak 60 jazyků. Je dostupný pro všechny běžně používané OS (MS Windows, Linux, MAC OS), čímž je vhodným pro jazyk Java pro zachování multiplatformního použití a je distribuován formou konzolové aplikace. [8]

Kladem systému je podpora více grafických formátů vstupních obrazů. Ačkoliv hlavním grafickým formátem pro aplikaci má být JPEG formát, aplikace je připravena pro

práci i s několika ostatními grafickými formáty. Formáty jsou následující: JPEG (JPG), PNG, GIF.

V aplikaci jsem implementoval vlastní metodu, která umí pracovat s Tesseract-OCR systémem. Podmínkou bezproblémové funkčnosti metody je přítomnost Tesseract-OCR systému v OS, přítomnost českých trénovacích dat a nastavení systémových cest v OS. Tyto podmínky jsou splněny standardní instalací Tesseract-OCR v uvedených OS. Instalace aplikace pro konkrétní OS uvádím v příloze uživatelské příručky. V případě nepřítomnosti OCR systému v OS je generována výjimka, která o této skutečnosti informuje. Metodu lze vidět ve výpisu číslo 4.

```
public String convertImageToStringOCR(String path) throws IOException,
        InterruptedException {
    if (isSelected)
        path = WORKING_SELECTION_FILE.getAbsolutePath();
    if (isThresholded) {
        path = WORKING_THRESHOLD_FILE.getAbsolutePath();
    }
    String result = "";
    if (path != null) {
        ProcessBuilder pb = new ProcessBuilder("tesseract", path,
                WORKING_TEXT_FILE.getAbsolutePath(), "-l", "ces");
        Process p = pb.start();
        p.waitFor();
        result = readTesseRactOCRResult(WORKING_TEXT_FILE.getAbsolutePath() + ".txt");
        new File(WORKING_TEXT_FILE.getAbsolutePath() + ".txt").delete();
    }
    setTextBuffer(result);
    return result;
}
```

Výpis 4: Volání metody pro použití OCR systému

Metoda je vytvořena tak, aby těsně spolupracovala s samotným OCR systémem, který běží v samostatném procesu. Metoda čeká na výsledek OCR systému a v případě selhání generuje výjimku. Aplikace není tak ohrožena v případě selhání OCR systému, nemůže dojít tudíž k pádu aplikace. Metoda je nastavena pro české prostředí, pracuje tedy s rozpoznáváním českých znaků. V případě budoucího rozšíření o jiné jazyky lze drobnou úpravou metody přidat podporu dalších jazyků.

Kombinací se segmentační metodou z výpisu číslo 1 lze dosáhnou kvalitnějších výsledků rozpoznání znaků. Celková doba trvání operace rozpoznávání znaku trvá v řádu sekund. Při aplikování segmentace obrazu a následného volání metody optické rozpoznávání znaků je narůst času rovněž v sekundách, což v celkovém součtu času nepřesahuje 15 vteřin u obrázků běžné velikosti digitální fotografie.

Jak jsem se již zmiňoval v části věnované OCR systémům, nelze nikdy zaručit 100 % úspěšnost rozpoznání textu v obraze. V aplikaci jsem tento problém zohlednil a umožnil jsem uživateli editaci extrahovaného textu a možnost jeho následného uložení do textového souboru. Taktéž jsem uživateli umožnil pomocí prvků uživatelského rozhraní použít filtraci extrahovaného textu. Pokud například budeme extrahovat řádek z fotogra-

fie jízdního řádu, lze se omezit pouze na rozpoznané numerické znaky a ostatní znaky ignorovat. Způsob filtrace textu je založen na práci s regulárními výrazy.

## 5.6 Audio syntéza

Při volbě vhodného audio syntetizéru jsem se rozhodoval mezi dvěma zástupci. Prvním zástupcem byl software vytvořený v rámci bakalářské práce panem Bc. Antonínem Brettšnajdrem, který se syntézou řeči ve své práci zabýval. Výsledkem jeho práce vznikl audio syntetizér napsaný v jazyku Java. Po otestování tohoto syntetizéru jsem dospěl k názoru, že není příliš vhodný pro mou práci, protože kvalita syntetizované řeči byla velmi nízká a po dotázání se nezávislých posluchačů jsem získal stejné zpětné ohlasy. Další nevýhodou tohoto syntetizéru byla nestabilita programu v případě syntézy většího množství souvislého textu.

Pro audio syntézu získaného textu jsem zvolil druhého zástupce a to syntetizér firmy Google nazvaný Google TTS. Tento syntetizér je součástí služby Google translator (Google překladač) a lze jej použít nezávisle na překladači. Výstupem audio syntetizéru je zvukový soubor ve formátu MP3. Důvody volby tohoto audio syntetizéru jsou následující:

- vysoká kvalita audio syntézy
- dostupnost zdarma
- podpora Českého jazyka

Jistým omezením syntetizéru je maximální množství vstupního textu. Ten je omezen na 100 znaků. Toto omezení jsem vyřešil implementací třídy *GoogleTTS*, která obsahuje metody pro správnou obsluhu syntetizéru a řeší vhodné dávkování delšího textu.

Syntetizér, jako takový funguje v online podobě, aplikace tak vyžaduje k provedení audio syntézy připojení k síti internet. To může znamenat jisté omezení v případě nedostupnosti připojení. Tento nedostatek v dnešní době nepředstavuje nepřekonatelnou bariéru, protože připojení k síti internet je dostupné více způsoby. Pokud budeme hovořit o použití aplikace na přenosném počítači, lze k připojení využít nabídek mobilních operátorů, kteří již poskytují za rozumnou cenu datové balíčky připojení k internetu. Další možností připojení k síti internet je využití připojení k síti Wi-Fi, pokud je v dosahu. Pokud se zaměříme na stolní počítače, tak tam je situace ještě lepší, protože dnes již existuje málo počítačů nepřipojených k síti internet. Je nutné poznamenat, že funkce audio syntézy v této aplikaci je hlavně určena uživatelům se zrakovým postižením a u těchto uživatelů není příliš předpokládáno, že budou aplikaci provozovat na mobilním počítači v terénu.

Jak jsem již výše uvedl, pro práci s audio syntetizérem jsem vytvořil třídu *GoogleTTS*, která řeší komunikaci s audio syntetizérem. Komunikace je řešená formou HTTP komunikace metodou GET. URL adresa Google syntetizéru je *http://translate.google.cz/translate_tts*. V tabulce číslo 5 jsou rozepsány jednotlivé parametry URL adresy.

Pro ukázku použití budeme chtít syntetizovat slovo „pokus". Vstupní kódování bude UTF-8, protože je to standardní kódování v HTTP komunikaci. Výsledný jazyk audio syntézy zvolíme češtinu. URL adresa HTTP požadavku tak bude vypadat následovně:

| Parametr | Popis parametru |
|---|---|
| ie | (input encoding) předpokládané kódování vstupního textu |
| q | (query) vstupní text pro syntézu |
| tl | (target language) výstupní jazyk audio syntézy |

Tabulka 5: Parametry audio syntetizéru

http://translate.google.cz/translate_tts?ie=UTF-8&q=pokus&tl=cs

Rychlost odezvy aplikace na požadavek audio syntézy je dána rychlostí odezvy serveru a kvalitou internetového připojení. Délka prodlevy bývá při zatížení serveru v řádu sekund. Prodleva mezi spojováním textových řetězců u textu delšího, než sto znaků je zanedbatelná v reálném použití. Tuto prodlevu lze přirovnat ke klesnutí hlasu v běžně mluvené řeči.

Aplikace využívá audio syntetizéru také k přehrávání dialogových oken s informacemi, které si uživatelé se zrakovým postižením nemohou plnohodnotně přečíst. Uživatelé jsou tak informování o dokončení operací OCR systému, dokončení segmentace obrazu, načtení souborů, stažení jízdního řádu a o ostatních dokončení operací. Tato funkce lze v případě potřeby vypnout v nastavení aplikace.

### 5.6.1 Přehrávání MP3 audio souborů

Jak jsem již výše uvedl, Google syntetizér vrací výstup audio syntézy v podobě souboru MP3. V jazyku Java pro přehrávání MP3 souborů využívám knihovnu JLayer. Tato knihovna je poskytovány pod licencí LGPL a je tak dostupná k volnému použití. Velkou předností knihovny je nízká zátěž procesoru. Procesor není zatížen ani z 1% při přehrávání MP3 souboru. Ukázku použití knihovny JLayer k přehrání MP3 audio souboru můžeme vidět ve výpisu číslo 5. [9]

```
public static void activatePlayer(InputStream medium) {
    Player player = null;
    try {
        player = new Player(medium);
        player.play();
    } catch (JavaLayerException e) {
        e.printStackTrace();
    } finally {
        if (player != null)
            player.close();
    }
}
```

Výpis 5: Ukázka použití knihovny JLayer

## 5.7 Jízdní řád MHD Ostrava

V průběhu tvorby bakalářské práce se ukázalo, že časté obrazy, ze který se snažíme pomocí systému optického rozpoznávání znaků získat text, jsou fotografie jízdních řádů. V aplikaci jsem proto implementoval sadu tříd, které spolupracují s informačním portálem IDOS a usnadňují tak přístup k těmto informacím.

Informační portál IDOS poskytuje informace o dopravě pro celou Českou republiku. Portál IDOS je provozován společností CHAPS s. r. o. , která poskytuje k práci s portálem IDOS několik programovacích API. Tato API jsou rozdělena na placená API pro širší škálu programovacích jazyků a také bezplatné internetové API pro komunikaci s portálem pomocí HTTP protokolu.[10]

V aplikaci jsem se rozhodl využít internetové API, protože je bezplatné a poskytuje aktualizované jízdní řády. Ovládání internetového API je popsáno v tabulce číslo 6, kde je pro získání vybraných informací o spoji nutné vyplnit vstupní parametry odpovídajícími hodnotami. Čas spojení je brán podle aktuálního času, nicméně ve výpisu lze vyčíst i časové údaje o následujících spojeních nebo o předcházejících spojeních.

| **vstupní parametr** | **popis parametru** |
|---|---|
| l | (link) číslo linky spoje |
| f | (from) z které zastávky |
| t | (to) do které zastávky |
| submit | potvrzení odeslání požadavku (hodnota true) |

Tabulka 6: Popis parametrů ovládání internetového IDOS API

Parametry jsou předávány do URL adresy a jsou odesílány metodou GET. Adresa má následující tvar:

http://www.idos.cz/*název města*/zjr/*parametry*

Pokud uvedeme příklad získání spojení ze zastávky Náměstí Republiky do zastávky Rektorát VŠB linkou číslo 8, tak tvar URL adresy HTTP GET požadavku bude následující:

http://www.idos.cz/ostrava/zjr/?l=8&f=namesti%20rep&t=rek&submit=true

Z ukázky lze vypozorovat, že není nutné zadávat celý název zastávky, ať již počáteční nebo koncové. Stačí zadat část názvu, kterou lze zastávku jednoznačně identifikovat. Aplikace také nedbá na striktní dodržování diakritiky a velkých nebo malých písem. Ve své aplikaci jsem se omezil na jízdní řády města Ostrava. Aplikace pracuje se všemi druhy dopravních prostředků MHD Ostrava, to znamená, že pomocí aplikace lze zjistit informace o přepravě pomocí tramvajových linek, trolejbusových linek a autobusových linek. Protože součástí internetového API je i prezentační vrstva, která není pro aplikaci využitelná, vytvořil jsem metodu, která na základě práce s regulárními výrazy oddělí užitné informace a dále je předá k zpracování dalším metodám. Ukázku metody pro získání časových údajů o zadaném spojení můžeme vidět ve výpisu číslo 6.

```
public List<TimeTableRow> getTimeTableRows(String rawData) {
    String patternString = "<td class=\"right bold\">(\\d+)</td><td class=\"zjrmin\">(.*?)</
        td>";
    Pattern pattern = Pattern.compile(patternString);
    Matcher matcher = pattern.matcher(rawData);
    ArrayList<TimeTableRow> list = new ArrayList<TimeTableRow>();
    while(matcher.find()) {
        TimeTableRow row = new TimeTableRow();
        row.setHour(matcher.group(1));
        String minutes = "";
        if (isWindows()) {
            minutes = matcher.group(2).replaceAll("(<img.*?/>)", "*").replaceAll("(&.*)", "");
        } else {
            minutes = matcher.group(2).replaceAll("(<img.*?/>)", "\u267f").replaceAll("(&.*)", "")
                ;
        }
        minutes = minutes.replaceAll("(V|Z)", "");
        row.setMinutes(minutes);
        list.add(row);
    }
    return list;
}
```

Výpis 6: Ukázka metody pro získání časových údajů o zadaném spojení

Protože informace o jízdních řádech hledají rovněž uživatelé s postižením pohybového ústrojí, jsou v aplikaci zobrazeny také informace o bezbariérových spojích.

Vyhledávání dopravních spojení v informačním portálu IDOS je v aplikaci úzce propojeno s audio syntetizérem a umožňuje tak zrakově postiženým uživatelům přečíst vybraný řádek jízdního řádu včetně informace o bezbariérovém spoji.

## 5.8 Práce s textovým soubory

Další funkcí aplikace je práce s textovými soubory ve více druzích formátů. Tato funkce byla vytvořena z důvodu využití audio syntetizéru i pro texty již uložené v textovém souboru. Aplikace není tak omezena pouze na text získaný výstupem OCR systému, ale umožňuje načíst i texty uložené z jiných datových zdrojů. Podporované formáty textových souborů jsou následující:

- **klasické textové soubory** - soubory s příponou txt vytvořené běžnými textovými editory daného OS
- **soubory ve formátu PDF** - soubory vytvořené specializovaným softwarem
- **soubory aplikace MS Word** - podpora souborů nejpoužívanějšího textového editoru firmy Microsoft, podporovány jsou soubory s příponou .doc a .docx
- **soubory ve formátu RTF** - RTF formát bývá najčastěji vytvořen aplikací MS WordPad

Pro práci se soubory ve formátu PDF jsem využil knihovnu PDFBox. Knihovna je poskytována formou otevřeného kódu (open-source) a je pod vlastnictvím organizace Apache Foundation, která knihovnu poskytuje pod licencí Apache License 2.0. Knihovna nabízí pro práci s PDF soubory spoustu funkcí, dokáže rovněž PDF soubory vytvářet nebo editovat. V práci využívám možnost přečtení PDF dokumentu.[11]

Pro zpracování souborů uložených ve formátu doc a docx jsem využil knihovnu Apache POI. Tato knihovna byla speciálně vyvinuta pro práci s dokumenty firmy Microsoft v jazyku Java, které jsou z převážné většiny vytvořeny kancelářským software MS Office. Vlastníkem knihovny je organizace Apache Foundation a poskytuje ji pod licencí Apache License 2.0.[12]

Častým formátem textových souborů je rovněž formát RTF. Tyto soubory bývají nejčastěji vytvořeny pomocí softwaru MS WordPad, který je běžnou součástí operačního systému Windows. Pro soubory uložené ve formátu RTF poskytuje jazyk Java podporu v rámci standardních knihoven. Ačkoliv standardní knihovny jazyku Java poskytují dobré možnosti pro práci s RTF dokumenty, existuje problém při zpracovávání vybraných českých znaků, které obsahují diakritiku. Tento problém jsem vyřešil implementací třídy *RTFCZEncoder*, která zmíněný problém řeší.

Zpracování textových souborů jsem sjednotil do jedné metody, která na základě přípony souboru určí, o jaký typ se jedná a použije pro práci s ním vhodnou třídu. Ukázku metody uvádím ve výpisu číslo 7.

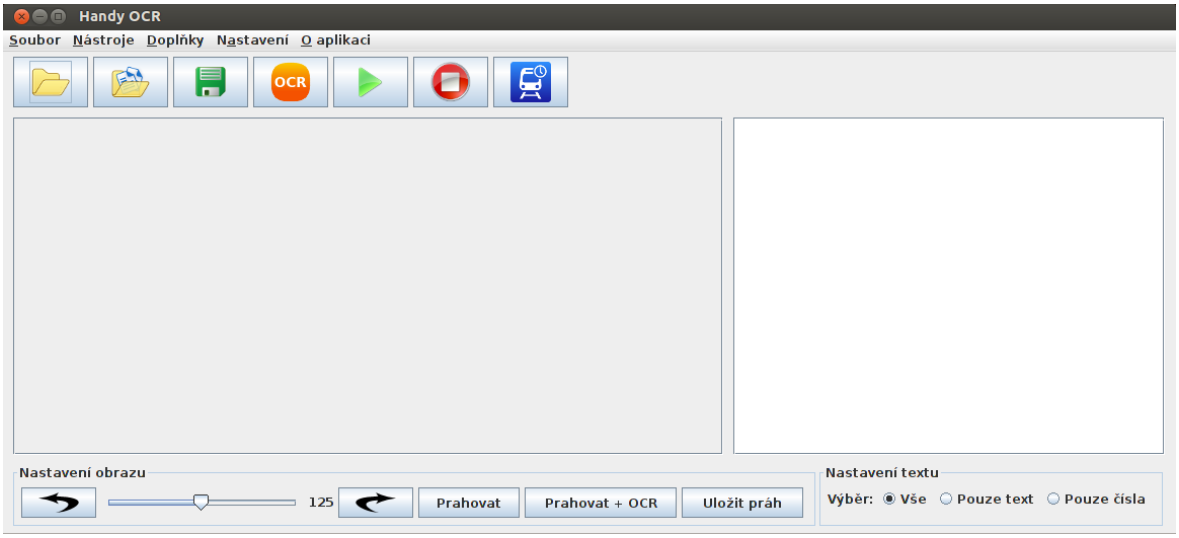
```
public static void activatePlayer(InputStream medium) {
    public String getTextFromFile(String path) throws IOException,
        BadLocationException, InvalidFormatException, OpenXML4JException,
        XmlException {
      File sourceFile = new File(path);
      String result = null;
      String fileExtension = Utilities.getExtension(sourceFile);
      if (fileExtension.equals(Utilities.rtf)) {
        result = RTFCZEncoder
            .encodeTextToRTFCZ(readTextFromFile(sourceFile));
        RTFEditorKit rtfKit = new RTFEditorKit();
        Document document = rtfKit.createDefaultDocument();
        rtfKit.read(new ByteArrayInputStream(result.getBytes()), document,
            0);
        result = RTFCZEncoder.decodeRTFCZToCzechText(document.getText(0,
            document.getLength()));
      } else if (fileExtension.equals(Utilities.pdf)) {
        PDDocument doc = PDDocument.load(sourceFile);
        PDFTextStripper stripper = new PDFTextStripper("UTF-8");
        result = stripper.getText(doc);
        doc.close();
      } else if (fileExtension.equals(Utilities.doc)
          || fileExtension.equals(Utilities.docx)) {
        POITextExtractor extractor = ExtractorFactory
            .createExtractor(sourceFile);
        result = extractor.getText().replaceAll("\\\n+", "\\\n");
      } else {
        result = readTextFromFile(sourceFile);
```

```
        }
        setTextBuffer( result );
        return result;
    }
```

Výpis 7: Metoda pro práci s textovými soubory

## 5.9 Uživatelské rozhraní

Uživatelské rozhraní aplikace bylo navrženo s ohledem na potřeby uživatelů se zrakovým postižením. Aplikace tak neobsahuje přebytečné ovládací prvky a vše je sjednoceno do jednoduchých tlačítek. Ukázku hlavní okna aplikace lze vidět na obrázku číslo 11.



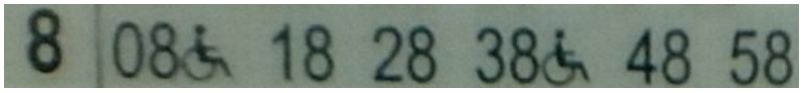

Obrázek 11: Ukázka hlavního okna aplikace

# 6 Zhodnocení dosažených výsledků

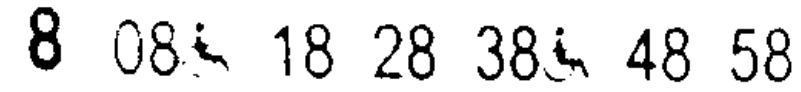
Aplikaci jsem úspěšně implementoval s veškerou uvedenou a předpokládanou funkcionalitou. Aplikace je rozložena do několika částí, kde každá část obstarává vybranou část celkové funkcionality. Tyto jednotlivé části bych v následujícím textu zhodnotil.

## 6.1 Segmentace obrazu - prahování obrazu

Segmentace obrazu přináší značné zlepšení rozpoznání znaků OCR systémem v případech, kdy je obraz velmi nekvalitní nebo má příliš nízké rozlišení (menší než 300 DPI). U obrazů, které jsou v dobré kvalitě a vyšším rozlišení (nad 300 DPI), není použití prahování vždy nutné, OCR systém dosahuje uspokojivých výsledků i bez něj. Po testování několika digitálních obrazů se ukázalo, že častým problémem bývá určení vhodné prahové hodnoty u nerovnoměrně osvětlených obrazů. U takových obrazů se vyplatí použít funkci pro oříznutí obrazu a rozpoznávat tak pouze vybrané části. Při jemném ladění vhodné prahové hodnoty u kvalitnějších obrazů lze dosáhnout 100% rozpoznání znaků v obrazu. Pro ukázku zlepšení kvality vstupního obrazu obsahujícího šum uvádím obrázky číslo 12 a 13.




Obrázek 12: Vstupní obraz před aplikací segmentační metody




Obrázek 13: Výstupní obraz po aplikaci segmentační metody

## 6.2 Optické rozpoznávání znaků

OCR systém Tesseract-OCR podává jedny z nejpřesnějších výsledků rozpoznávání textu z volně šiřitelných OCR systémů. Během testování se ukázalo, že přesnost rozpoznávání klesá u obrazů s rozlišením menším jak 300 DPI a také u rozpoznávání písem, které jsou menší, jak 10 bodů. U velmi nekvalitních obrazů lze výsledek OCR systému zkvalitnit použitím metod výše uvedené segmentace obrazu s vhodným ořezem obrazu. I po aplikování ořezu či segmentace obrazu je u velmi nekvalitních obrazů rozpoznání textu na nízké úrovni. V takových případech je již nutné text ručně opravit, případně dopsat. Mezi testovanými obrazy se nacházely i obrazy obsahující nestandardní znaky, jako je například znak vozíčkáře ve fotografii jízdního řádu. Tyto znaky jsou všeobecně OCR systémy hůře rozpoznatelné, protože jsou velmi málo používány, jako vzorová data při fázi tréningu OCR systémů. Tesseract-OCR se snaží tyto speciální znaky rozpoznat, jako

klasické písmeno. Pro řešení tohoto problému lze v aplikaci použít selekci textu, kterou lze v případě rozpoznávání řádků jízdního řádu odstranit alfabetické znaky a vytvořit tak pouze numerický text.

Rychlost rozpoznávání znaků je dobrá, u rozměrově větších obrazů (okolo 2500x2000 px) dochází k prodlevě v řádu jednotek sekund. U rozměrově malých obrazů je prodleva zanedbatelná.

Souhrnně lze pak vyhodnotit rozpoznávání znaků v aplikaci za vyhovující a v praxi použitelné.

## 6.3 Audio syntéza

Zvolený audio syntetizér od firmy Google potřebuje ke své funkčnosti připojení k síti internet. Omezením zde může být rychlost reakce serveru na požadavky, či kvalita síťového připojení. Co se týče kvality odezvy serverů firmy Google, tak ta se během testování ukázala jako výborná. Testování probíhalo na připojení pomocí Wi-Fi rychlostí 1 Mbit/s. Pokud by došlo k připojení pomocí služeb mobilního operátora, mohl by nastat problém v případě připojení pomocí GPRS. V případě připojení pomocí sítě 3G je tento problém nepravděpodobný, protože takové připojení svou rychlostí převyšuje uvedenou testovanou rychlost připojení pomocí Wi-Fi.

Audio syntetizér je v aplikaci využíván nejen k čtení extrahovaného textu, ale také ke čtení informačních dialogů a zpráv, které slouží pro informování zrakově postižených uživatelů. Zde rovněž syntetizér reaguje velice rychle a uživatel je zvukově informován ihned po vyvolání libovolné události.

Omezení, na maximální počet vstupních znaků (100), které má použitý syntetizér vedlo k rozdělení delšího souvislého textu na části, které jsou postupně dávkovány syntetizéru. Toto omezení se v průběhu syntézy delších textů výrazně neprojevuje, pomlky, které vznikají v částech rozdělení jsou velmi krátké a pro uživatele působí, jako pokles hlasu při ukončení věty.

# 7 Závěr

V práci jsem se zabýval problematikou získávání textových informací z digitálně pořízených obrazů a následnou reprodukcí získaných informací pomocí audio syntézy. K řešení dané problematiky jsem navrhl a realizoval aplikaci, která je svým návrhem uživatelského rozhraní vhodná i pro zrakově postižené uživatele.

V první části práce jsem analyzoval segmentační metody vhodné pro předzpracování vstupního obrazu, které obraz upraví do vhodné podoby pro následnou metodu optického rozpoznávání znaků. Část druhou jsem věnoval analýze a seznámení s principem metody optického rozpoznávání znaků známou také pod zkratkou OCR. Uvedl jsem zde základní typy metod optického rozpoznávání znaků a zhodnotil jsem současný stav OCR systémů. V následující části jsem se seznámil s problematikou audio syntézy textu na počítači a jejím spojením s lidským způsobem tvorby řeči.

V aplikaci jsem využil vybraný OCR systém, který jsem upravil do vhodné podoby pro snadné použití. V návaznosti na získaný textu z OCR systému jsem zvolil vybraný audio syntetizér pro provedení audio syntézy, který jsem rovněž upravil do vhodné podoby a také jsem jej využil pro komunikaci aplikace se zrakově postiženými uživateli. Aplikace byla oproti původnímu návrhu rozšířena o práci s jízdními řády MHD Ostrava tak, aby využitím audio syntetizéru byla zrakově postiženému uživateli co nejvíce přínosná čtením jednotlivých řádků jízdního řádu.

Jako budoucí vylepšení aplikace bych viděl přímé začlenění OCR systému do aplikace využitím JNI a kompilací OCR systému do dynamické knihovny (DLL, SO) pro vybrané platformy (MS Windows, Linux, MAC OS). Výsledkem by byla snazší distribuce aplikace v podobě jediného souboru (JAR archívu). Dalším vhodným vylepšením by bylo přidání více filtrů pro segmentaci obrazu, případně přidání nových typů segmentačních metod.

# 8 Reference


[1] KALOVÁ, Ilona. *Segmentace a detekce geometrických primitiv* [online]. [cit. 20. 2. 2013]. Dostupné na WWW: http://www.uamt.feec.vutbr.cz/vision/TEACHING/MPOV/05%20-%20Segmentace%20a%20detekce%20geometrickych%20primitiv.pdf

[2] LIČEV, Lačezar. *Fotogrammetrické systémy a rozpoznávání zájmových bodů* [online]. [cit. 21. 2. 2013]. Dostupné na WWW: http://gis.vsb.cz/GIS_Ostrava/GIS_Ova_2001/Sbornik/Referaty/Licev1r.htm

[3] VYMETÁLEK, Petr; VIKTORA, Jan. *OCR - metoda optického rozpoznávání znaků* [online]. [cit. 8. 3. 2013]. Dostupné na WWW: http://geo3.fsv.cvut.cz/vyuka/kapr/SP/2008_2009/vymetalek_viktora/semestralni_prace.pdf

[4] ČERNOCKÝ, Jan. *Zpracování řečových signálů* [online]. [cit. 17. 3. 2013]. Dostupné na WWW: http://www.fit.vutbr.cz/study/courses/ZRE/public/opora/zre_opora.pdf

[5] PSUTKA, Josef. *Komunikace s počítačem mluvenou řečí* Praha:Academia, 1995. 287 s. ISBN 80-200-0203-0

[6] PRÁŠEK, Petr. *Formantová syntéza* [online]. [cit. 24. 3. 2013]. Dostupné na WWW: http://noel.feld.cvut.cz/vyu/dzr/dzr12/

[7] JELÍNEK, Lukáš. *Java (24) - úvod do grafiky a GUI* [online]. [cit. 27. 3. 2013]. Dostupné na WWW: http://www.linuxsoft.cz/article.php?id_article=1184

[8] GOOGLE Inc. *tesseract-ocr* [online]. [cit. 28. 3. 2013]. Dostupné na WWW: http://code.google.com/p/tesseract-ocr/

[9] JZOOM. *JLayer - MP3 Library* [online]. [cit. 28. 3. 2013]. Dostupné na WWW: http://www.javazoom.net/javalayer/javalayer.html

[10] CHAPS s. r. o. . *IDOS Řešení pro internet* [online]. [cit. 29. 3. 2013]. Dostupné na WWW: http://www.chaps.cz/files/idos/IDOS-API.pdf

[11] THE APACHE SOFTWARE FOUNDATION. *Apache PDFBox - Java PDF Library* [online]. [cit. 30. 3. 2013]. Dostupné na WWW: http://pdfbox.apache.org/

[12] THE APACHE SOFTWARE FOUNDATION. *Apache POI - the Java API for Microsoft Documents* [online]. [cit. 30. 3. 2013]. Dostupné na WWW: http://poi.apache.org/

# A Přílohy na přiloženém CD

## A.1 Příloha 1

Text této bakalářské práce v elektronické podobě

## A.2 Příloha 2

Zdrojové kódy aplikace + spustitelná aplikace s instalačním skriptem

## A.3 Příloha 3

Uživatelská příručka

## A.4 Příloha 4

Programátorská dokumentace ve formě JavaDocu